Chapter 1
Concept of Programming Language

1.1 Introduction

Microprocessor or μp is the most important part of a computer. With the help of devices and chips, a microprocessor can accept instructions and data. A μp can manipulate this data to make decisions. The step-by-step instructions based on logics to solve a problem is a computer programme. The manufacturer will incorporate the circuits and elements of the microprocessor so that it can perform jobs. Each job is designated with a number in the binary format. So, to use a microprocessor we must feed these numbers. A set of instructions written with these numbers and address of memory location of operands (Data) is called a ***Machine Language Instruction.***

To solve a problem, we must develop a detailed and precise step by step method. These steps are called ***Algorithm or Logical development of the programme***. If it is made in a pictorial format, it is called as ***flow chart***. Each step in this algorithm can be decoded to machine language instructions. This sequence of machine language instructions is known as ***Machine Language Programme***.

A machine language programme depends on the type of microprocessor. Due to the machine dependency of the language and the difficulty to handle the numbers, debugging become a difficult task. To avoid this, another language is developed. In this, the instructions are some combination of two or three characters that have a close resemblance to the corresponding English word for that job. MOV, ADD, HLT etc. are examples. A programme with this type of instructions is called ***Assembly Language***. A programme on the computer can convert the Assembly Language to Machine Language for the working of a microprocessor. This programme for translation is known as ***Assembler.***

1.2 Algorithm

In computer science, an algorithm usually means a small procedure that solves the problem. It is a procedure or formula for solving a problem, by a sequence of specified actions. A computer program is an elaboration of this algorithm.

An algorithm must have the following characteristics:

- The algorithm should be clear and unambiguous. Each of its steps should be clear in all aspects and must lead to only one meaning.
- If an algorithm says to take inputs, it should be well-defined inputs.
- The algorithm must clearly define what output should be made and it should be well-defined as well.

- The algorithm must be finite, i.e., it should not end up in an infinite loop or similar.
- The algorithm must be simple, generic, and practical, such that it can be executed with the available resources. It must not contain some future technology or anything like that.
- The Algorithm designed must be language-independent, it must be plain instructions that can be implemented in any language, and the output will be the same.

Example 1: Write an algorithm to add three numbers and print the sum.

Hints: - Before writing the algorithm analyse the question and possible methods of solution like following

1. The problem is to read 3 numbers and to print their sum.
2. The input must be 3 integer numbers.
3. Each input data must have some name to identify them. Say num1, num2 and num3.
4. There must be some properly named variable to store the answer. Say sm for sum.
5. The output must be the sum of the three numbers taken as the input. So, sm = num1 + num2 + num3
6. The output is to be printed.

Algorithm:

Step 1: START
Step 2: Declare 3 integer variables num1, num2 and num3.
Step 3: Take the three numbers, as inputs in variables num1, num2, and num3, respectively.
Step 4: Declare an integer variable 'sm' to store the resultant sum of the 3 numbers.
Step 5: sm = num1 + num2 + num3
Step 6: Print the value of variable sm
Step 7: END

Example 2: Write an algorithm to find the smallest number from the given two numbers.

Hints: - First we must input two integer numbers. So, we need two names(variables) to identify them. Let they are num1 and num2. If num1< num2 print num1 as the smallest else print num2 as the smallest.

Algorithm:

Step 1: START
Step 2: Declare 2 integer variables num1 and num2
Step 3: Take the two numbers, as inputs in variables num1 and num2 respectively.
Step 4: If num1 < num2 print num1 as the smallest else print num2 as the smallest
Step 5: End

1.3 High Level Language

Writing a programme in Machine Language or Assembly Language is tiresome. It needs a deep understanding of hardware and memorising the operational codes. Due to these demerits, many other languages have been developed to use in a computer. They are generally known as ***High Level Languages***. Ordinary human languages cannot be used for computer due to their confusing grammar rules, vocabulary with synonyms and phrases, colloquial variations, Speed & contest dependency of meaning etc. A High-Level Language has a close resemblance to the English language. It is with a set of well-defined keywords and usage rules to make a direct translation of the Algorithm easy. BASIC, FORTRAN, C++, PYTHON etc. are the examples. In this, each instruction is with specific rules of usage. It can be used in any computer regardless of the brand and manufacturer of the microprocessor. So, this is a machine independent language.

The usage of each instruction or keyword is confined to rigid rules. Only if we write the instructions within these rules, the computer can understand them. These rules like grammar rules in English are known as ***Syntax Rules***. An instruction even if correct in syntax, may lead to wrong answers if it is not properly used. That is, the instruction is with ***Semantic Error***. For instance, consider the sentence 'Mary plays the violin'. Hereby human practice, we know that the name of the child is 'Mary' and the name of the instrument is 'violin'. Suppose I made a special type of musical instrument and named it 'Mary' in memory of somebody. My daughter named 'Veena' is an expert in this instrument. When she plays the new instrument, I have to say – 'Veena plays Mary'. Then the listener may be confused about the name of the child and the name of the instrument. Some of them will say that there is a grammar mistake. Here the statement is with lesser clarity. This is an example of a semantic error. If I am writing the above statement as – 'My daughter Veena is playing the new instrument named Mary', it is free from syntax errors and semantic errors, even if it is little lengthy. Computer codes must be free from syntax error and semantic error.

A high-level language must have the following features.

1. Well defined set of permitted characters.

2. Ability to represent and manipulate different data types like integer, float, character etc. and objects like strings, arrays, list etc.
3. A detailed list of operators (arithmetic, conditional etc.) to act on the data.
4. Instructions for control flow on decision making, branching, looping etc.
5. A set of syntax rules to frame the correct instruction with keywords and symbols permitted by the language.
6. A set of semantic rules to assign correct and unambiguous meaning to a statement.

1.4 Compiler and Interpreter

A programme written in a high-level language is called ***Source Code***. We can feed this programme directly to a computer using any text editor. But microprocessor can understand only machine language. So, there will be an elaborate computer programme to translate the source code to machine language. There are two types of translator programme available now. They are ***Interpreter and Compiler***.

An interpreter reads and executes the high-level programme, line by line. During the execution, if a line is syntactically wrong, it stops the execution with an error message. After correcting it, again we must start from the beginning. During all executions of the same programme, the computer repeats the same steps, which is highly time consuming.

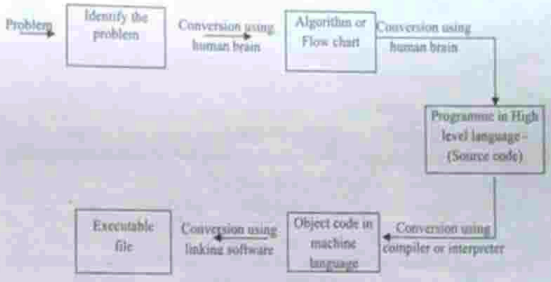
The mode of operation of a compiler is entirely different. Instead of executing an instruction, the compiler converts the entire programme into machine language and stores it to a file. During compilation, the compiler will go through the entire programme and will list out all possible errors with the line numbers. We can correct it and compile it again. If the programme is free from syntax errors, the compiler will convert the programme to machine language and stores it in a location. We can directly execute the programme from this stored file. So, we need the compilation only once. More than that the original source code is safe in the hands of the programmer.

1.5 Development of a computer programme

Writing a programme involves the following steps.

1. Study the problem and develop the logic and algorithm with minimum steps and minimum complexity to get the correct solution.
2. Select the language with which we are familiar.
3. Translate the algorithm to the selected language with correct notations, syntax rule and semantic rule. This programme is known as ***Source Code.***

4. Convert it into machine code using the proper compiler or interpreter. Some language uses the interpreter whereas some language uses the compiler.



Detailed steps in the development of a programme

5. After compilation computer will develop a file named object code, which is in machine language.
6. After linking the object code with libraries and other external segments object code is converted into another file named ***executable file***. Store this executable file in the proper place. To run the programme, we must invoke this file only. No need for source code or compiler.

Exercise

One-word type questions

1. Conversion of an algorithm to source code is done by----------
2. Conversion of source code to object code is done by ------
3. Conversion of object code to exe file is done by ---------
4. The stage in between the source code and executable file is ---------

Short answer type questions

1. What is a machine-language instruction?
2. What is an assembler?
3. What is a high-level language? Give example?
4. What are the major defects of assembly language?

5. What is the meaning of the syntax rule?
6. What is the meaning of semantic rule?
7. Explain the role of a microprocessor in a computer.
8. Explain the major disadvantage of Machine Language which lead to the introduction of assembly language.
9. Distinguish between compiler and interpreter.

Paragraph type questions

1. What is an algorithm? Write its essential features.
2. Explain the steps in the conversion of a problem to an executable programme.
3. What are the features of high-level language?
4. Write an algorithm to read two numbers and print the largest number.

Long Answer type questions

1. Explain the steps involved in the evaluation of different types of computer languages, with the merits and demerits of each phase.

Chapter 2
Starting with Python

2.1 Introduction

The implementation of Python was in December 1989 by a team headed by Guido Van Rossum of the Netherlands. Rossum was a fan of Monty Python's Flying Circus, a famous British comedy programme of that time. He adopted the name 'Python' from this sketch for his new language. In 2000 Python got the present scenario with some major changes like shifting to more transparency and community link under free software movement. ***Free software*** can be used, studied, and modified without any restriction. It can be copied and redistributed in modified or unmodified form. Free software is generally available without charge. Software becomes free software only if the human-readable form of the program (Source code) is made available to the recipient along with a notice granting the permissions to use, study and modify. The free software movement was conceived in 1983 by Richard Stallman to give the benefit of "software freedom" to computer users. Stallman founded the Free Software Foundation in 1985 to provide the organizational structure to advance his Free Software ideas. Later, alternative movements like 'Open-Source Software' started. Python is a software in this category. So many experts in the computer industry made many remarkable changes to this software. Python contains aspects of philosophy influenced by many experts who shifted to free software movement from popular languages like Java, Perl etc. As a result, Python becomes more and more powerful.

2.2 Advantages of Python over other languages

1. It is a general-purpose, high-level programming language with an emphasis on code readability.
2. Python is in the category of 'Open-Source Software' and so no need of paying any licence fee to use it. Its source code is freely available. So, everybody can modify it, to make it better.
3. It is powerful with clear syntax.
4. It is so abbreviated due to its power to couple the instructions together.
5. The standard library is large and comprehensive.
6. Since it is an open-source software, many experts from different part of the world will contribute many functions to the library. So, the library will expand day by day.
7. The use of indentation for block delimiters (will explain later) is unique among the most popular languages. Due to this debugging is easier.

8. The "type system" of Python is dynamic (will explain later).
9. It offers strong support for integration with other popular languages and tools in the market.

2.3 Interactive Mode and Script Mode (File mode)

Python can be used in two modes. Interactive mode and script mode. Interactive mode can be termed as calculator mode or shell mode due to its nature of the operation.

If we are opening Python without any arguments in the command prompt of the Operating System, the interpreter will be active with a display '>>>'. This is called ***Command Prompt***. Type a single python statement on this prompt and press the enter key. The system will show the answer if there is an answer to display. Otherwise, the prompt will appear again. This interaction can be continued. A complicated mathematical expression also will work. So, this method is called ***Interactive Mode***. This can be used as a calculator to view the result of simple or complicated calculations including complex numbers and even to view graph. We can type an arithmetic expression at the prompt to get the answer. The operators +, -, x and / will work like a calculator. So, this mode can be called as ***calculator mode.*** For example, type "x=10" on the command prompt and press enter key. Since nothing to display as the answer, the command prompt will appear again. Type "y=5" and press enter key. Again, the command prompt will appear. Type "print x*y" and press enter key. Just below the prompt, the system will show the answer. The command prompt will appear again. In python documentation, the above example can be written as follows.

```
>>> x=10↵
>>>y=5↵
>>>print (x*y)↵
50
>>>
```

In the calculator mode, we can use python without variables.

```
>>> 1238*32↵
39616
>>>
```

Parentheses can be used for grouping.

```
>>> (50-5*6)/4)↵
5
```



```
>>>(1+2j)/(1+1j)↵
(1.5+0.5j)
```

But this method cannot be used for large and complicated works involving loops, iterations, and conditional statements. ***Script Mode*** is the preferred way in such cases. In this mode, type the source code using any available text editor like notepad in Windows, or idle in Python Shell or gedit in Linux. The filename must be with an extension '.py'. This can be compiled and operated by Python. Details of installation and operational steps are included in the appendix.

2.4 Naming rules

Any alphanumeric character combination is permitted.

1. There must not be any white space in the name.
2. Underscore ('_') is permitted.
3. Extension must be '.py' always.

2.5 Python Programme layout

An ordinary python programme contains the following elements. All the elements are not mandatory but depend on the problem it may present.

1. Import statements to activate the relevant part of the translator to RAM.
2. User made functions and declarations.
3. Inputs
4. Settings for graphics and file operations
5. Calculations, decisions, and loops
6. Outputs

```
# programme to find the range of a projectile
from math import *
vel,ang=input('Enter the initial velocity and angle of projection ')
rad_ang=ang*3.14/180
rang=vel*vel* sin(2*rad_ang)/9.8
if ang==45:
    print 'The range is maximum'
print( 'The range is %8.4f' %(rang))
print ('Thank you')
# end
```

A sample programme

In the example illustrated in figure, the first line is the title of the programme for easier understanding, which is not compulsory. Anything written after the hash symbol ('#') will be ignored by the translator. So, it can be used by the programmer to write titles, explanatory notes etc. In the next line, we are loading a specific part of the translating software names 'math' to RAM. The mathematical functions such as Sine, Cosine, Absolute value etc. are interpreted in this module. The next line is to take two inputs from the keyboard, which is necessary for the calculation. Each input to a computer must have a name (Called as an identifier). Here 'val' and 'ang' are the identifiers for these two inputs. The next two lines are the calculation part. In the next line, we are making a conditional operation. The condition is that whether ang=45 or not. If the condition is true, then only, the computer will execute the next commands written with an indent. (Indent means, writing after a definite space). Then the control jumps to the next and executes the print statements. The last sentence is a note for the reader about the end of the programme, which is not a must.

2.6 Variable (Identifier)

In a programme, we must use many data, like values, constants, functions, arrays, lists etc. In a computer programme, each data must have a name. Names using for this purpose is known as ***variable*** or ***identifier***. Consider the following equations.

a=b+c

*per=mark * 100/max_mark*

In the above equations a, b, c, per, mark and max_mark are the examples for variables.

Rules of naming the variable.

1. Any alphanumeric character combination is permitted.
2. A variable name must start with an alphabet.
3. White space is not permitted in the name.
4. Underscore ('_') is permitted.
5. Variables are case sensitive. That means for a computer 'mark' and 'Mark' are two different quantities.
6. The permitted length of a variable name depends on the microprocessor type.
7. Even though any character combination can be used as a variable name, better to use some meaningful combinations of English alphabets for effective programming. Eg: - mark, reg_no, ph_mark etc.

8. Keyword (the words using in an instruction command) cannot be used as a variable name.

Valid variable names	Invalid variable names
mark	def
mark1	1mark
mark_phy	Mark-phy
no	No.
m2k3	Roll no
kk_tho	Else

2.7 Input statements

Input statements are the instructions in python to input data from the keyboard to the computer. The most common formats of input statements are the following.

mark = input()

mark = input('Enter the mark ')

When this statement is executed, the cursor will wait for an entry from the keyboard. User must know what to do when the cursor blinks during the execution of the programme. Giving a message on the message board is the better way of solving it.

mark = input('Enter the mark')

We can receive more than one data using a single input instruction using split function.

x,y = input ('Enter the x-value and y-value').split()

The general syntax of these instructions are the following,

identifier = input (*message within quotes which is optional*)

identifier list separated by comma = input (*message within quotes which is optional*).split()

The number of variables in the list must match with the number of data typed through the keyboard. During data entry, data must be separated by space.

In interactive mode the allocation of data type for each variable is automatic. As we type python recognize the data type, means whether it is float or integer or complex etc. But in script mode, all data is received as string, which we can convert to any type by casting operators, *like x=float(y), x=int(x).* Anyhow, there is no need of variable declarations at the starting. So, we can say that Python is a ***dynamic data type*** language.

In the older versions (2.x) there was an another type of input in which, the computer will accept the data without any type of specification.

For example, *x,y=raw_input("enter x and y values ")*. After this instruction, we can convert the data into any data type. If we are not specifying the type, the computer will treat it as a string.

2.8 Output statements

During the execution of a programme, the computer must show some messages and answers on the screen. We can use the following patterns of output instructions to display a text or numeral on the screen.

```
print(x)
print(no, mark)
print('Hello')
print('The answers are', x,y)
print((x+3)/y)
print('The sum of ', x, ' and ', y, ' is ', x+y)
```

In the above examples, we can see how the print statement is using along with variables, strings or with a combination of strings and variables. Try to understand the mixing of messages and variables separated by a comma. There is an another method called formatted output which will be discussed later. Each print command will force the computer to start the printing in a new line. If we want to print the next instruction in the same line, we have to add an instruction 'end=' '

```
print ('Hello', end=' ')
print ('Thomas')
```

Output will be 'Hello Thomas'. The same can be done with slash operator '\n' ('n' for new line)

```
print('hello \n Thomas')
```

Output will be,

```
'Hello'
'Thomas'
```

If we want spaces in between the words or values(Just like matrix printing) we can do it as follows

```
print('A', 'B', 'C', sep='    ')
```

Output will be,

```
A    B    C
```

The same can be done with slash operator '\t' (t for tab). Go through the following python programme to demonstrate more I/O instructions.

Example 1: io.py

```
# Programme to demonstrate input and output instructions
x=input('Enter an integer')
x=int(x)
print ('x = ',x)
print (type(x))
print (float(x))
print (complex(x))
y=input ('enter a name')
print (y)
print( y+' is a good boy')
#end
```

Output

```
Enter an integer 14↵
x =  14
<type 'int'>
14.0
(14+0j)
enter a name Babu↵
Babu
Babu is a good boy
```

Example 2: io1.py

```
x='I am an'
y='Indian'
print('Single line print')
print (x, end=' ')
print (y)
print('New line print')
print ('I am\n an\n Indian')
```

Output

```
Single line print
I am an Indian
New line print
```

I am
an
Indian

2.9 Case sensitivity

Python is a case sensitive language. Most of the commands and keywords are defined in lowercase letters. So, we must use lowercase itself for such keywords. The variables can be in uppercase or lowercase. But the variables assigned in lower case and uppercase are distinct. In python, Mark and mark are different.

2.10 Data types

Before proceeding with computational work, the computer must know the exact nature of the data. This is to save time and memory space. Depends on the nature of the operation, python can deal with the following data types.

1 Numeric
2 Sequence
3 Set
4 Dictionary

The numeric data types are integer, float, and complex. Integer type means a plain integer without a decimal part. The length of the integer depends on the microprocessor. Python language cannot deal with an integer above this limit. Float represents the decimal number with 15-digit accuracy. The numbers higher than this will automatically shift to exponential form. Complex numbers can deal with a real part and an imaginary part.

String, Tuple and List are the data types coming under the category sequences. We will discuss it later.

Rules of type conversion

1. During addition and subtraction, if both are integer, the result will be an integer. If anyone or both is a float, the result will be a float.
2. During division or multiplication, if both operands are integer, the result will be a truncated integer.
3. If anyone operand is a float and the other is a float or integer, the result will be a float.

During the programme, we must give extra care to deal with the data type. To solve such situations, we must convert one data type to another depends on the situations. The instructions for this are known as ***data type casting instructions***.

To convert an integer 'per' to a float, we can use *per=float (per)*

Similarly, the instruction, *no=int(no)* can be used to convert a float to an integer. This can be done by any one of the following instructions. Go through the following different instructions.

rad=int(input("Enter the radius of an atom")
rad=float(input("Enter the radius of an atom")

List of type casting instructions

1 *y=float(x):* Convert an integer to float
2 *y=int(x):* Convert a float to integer by truncation.
3 *z=complex(x,y):* Create a complex number with x as real part and y as imaginary part If *y* is omitted, it defaults to zero
x=2
y=3
z=complex(x,y)
Then z=(2+3j)
4 *str(x):* Convert a number to a string. The output will not support any mathematical functions.
x=2.75
y=str(x)
Then value stored for y will be='2.75'. It does not have any property of a number. The value stored for y +y will be='2.752.75' due to string concatenation
The instruction *'print (type(x))'* is to know the data type of a quantity represented by x. Go through the following python programme to demonstrate more string activities.

Example 3: types.py

```
#programme to demonstrate input and output data types
x=10
print('x = ',x)
print('x is a ', type(x))
y=x/3
print('y= ', y)
print('y is a ', type(y))
z=float(x/3)
print('z= ', z)
print('z is a ', type(z))
```

```
s='Hello, I am python boy'
print(s)
print('Now s is a ', type(s))
c=2+3j
print('c =',c)
print('Now c is a ', type(c))
#end
```

Running the programme str.py will generate the following output.

```
x =  10
x is a  <class 'int'>
y=  3.3333333333333335
y is a  <class 'float'>
z=  3.3333333333333335
z is a  <class 'float'>
Hello, I am python boy
Now s is a  <class 'str'>
c = (2+3j)
Now c is a  <class 'complex'>
```

2.11 Assignment statement

An assignment statement is a statement to assign a variable name to a data. '=' is the assignment operator. Go through the following examples.

x=10

In this, we are assigning a value of 10 for the identifier x.

x=x+4

In this, we are asking the computer to change the present value of x in memory with x+4. It is to be remembered that the RHS may be a value or string or a mathematical expression. RHS must be a variable name. x=y+10 is valid whereas y+10=x is not a valid assignment. Python supports chain assignment also. x=y=z=12 is a valid ***chain assignment***. A string also can be assigned to a variable.

```
nam1='ANTONY'
name2='Thomas'
nam=nam1+nam2
```

In the assignment, *per=mark*100/max_mark*, RHS is a valid expression, and the result is assigned to the variable per.

2.12 Strings

Any collection of alphabets, digits, special characters, and white space is known as a ***character string*** in a computer language. Any quantity assigned within the double quotes or single quotes for an input statement will be a string. Any undeclared quantity received by raw_input also will be a string. The string addition, multiplication etc are quite different from numerical addition and multiplication. Strings can be used to store names, information, status etc. Go through the following examples for stings.

```
x='hello'
a='Thomas is a good boy'
p='' Pin code -680 001''
no='1002'
```

In string manipulation, the following instructions are common:

```
a='Thomas'
b=' is a good boy'
x=a+b
```

Then the value stored for x will be ' Thomas is a good boy '. The instruction x=3*a will assign "ThomasThomasThomas" for x. This is concepts are called string concatenations. Go through the following python programme to demonstrate more string activities.

Example 4: str.py

```
#example to demonstrate string activities
s1='I am'
s2='Babu'
s3='an'
s4='Indian'
print ('hello')
print (s1+s2)
print (s1+' '+s2)
print (s1+' '+s4)
#end
```

Running the programme str.py will generate the following output.

```
hello
I amBabu
```

I am Babu

I am Indian

Example 5: str1.py

```
#example to demonstrate string activities
print(s4)
print(3*s4)
print(s4[0:3])
print(s4[0:4])
print(s4[0:5])
print(s4[2:])
print( s4[:3])
#end
```

Running the programme str.py will generate the following output.

Indian

IndianIndianIndian

Ind

Indi

India

dian

Ind

Extracting a part of a string from the original using the above instructions is known as ***slicing.***

2.13 Operators

Operators are tokens that trigger some computation when applied to variables or elements of expressions. Python supports many operators to do computational works. Operators are divided into two.

1. Binary operator: They are the operators that require two operands on the left and right of the operator. Eg: +, -, x etc.
2. Unary operator: They are the operators that require only one operand to operate upon. Eg: +, -

The following list gives a brief description of commonly using operators and their functions.

Arithmetic operators

Exponentiation			
Symbol **	Example 3**2	Result 8	To find 3^2
Multiplication			
Symbol *	Example 3*2	Result 6	Ordinary multiplication. If any one of the operands is a float, the result will be float. If both are integer, the result will be an integer
Division			
Symbol /	Example 4.8/2 4/3	Result 2.4 1.33...	Ordinary division. The result will be a float always. If both are integers, then also result will be a float
Floor division			
Symbol //	Example 4.4//2 7//2	Result 2.0 3	A special type of division to get the quotient. If any one of the operands is a float, the result will be float. If both are integer, the result will be an integer. In both cases, the result will be a truncated one to the lowest whole number.
Modulus division			
Symbol %	Example 4.4%2 7%2	Result 0.4 1	A special type of division to get the reminder. If any one of the operands is a float, the result will be float. If both are integer, the result will be an integer. In both cases, the result will be the reminder after getting a quotient of the whole number
Addition			
Symbol +	Example 10+8 10+8.0	Result 18 18.0	Ordinary addition. If any one of the operands is a float, the result will be float. If both are integer, the result will be an integer

Subtraction			
Symbol	Example	Result	Ordinary subtraction. If any one of the operands is a float, the result will be float. If both are integer, the result will be an integer
-	10-8 10-8.0	2 2.0	

Negation			
Symbol	Example	Result	The value stored in x reverses the sign. It is a unary operator.
-	-x	-x	

Relational operators

These operators are binary operators, to check wheatear a mathematical condition is true or false. This is to make a decision in 'while loop', 'if loop' etc. Always the result will be Boolean.

Name of operator	symbol	Example	Result
Less than	<	2<3	.T.
Less than or equal	<=	3<=2	.F.
Greater than	>	3>2	.T.
Greater than or equal	>=	5>= 5	.T.
Not equal	!=	6 !=6	.F.
Equal	==	12==12	.T.

We must give extra care on using conditional equal to during a conditional check. If the variables involved are of integer type, exactness is possible. If it is a float, a perfect exactness is impossible in most case. It is due to the difference in the accuracy level of the microprocessor.

Logical operators

These operators are binary operators, to check wheatear more than one mathematical condition is true or false. This is to make a decision in 'while loop', 'if loop' etc with multiple conditions.

Name of operator	Symbol	Example	Result
Logical OR	or	x=2 or y=4	True if x=2 or y=4
Logical AND	and	x=2 and y=4	True only if x=2 or y=4
Logical NOT	not	not y>9	It is a unary operator. True only if y>9 is false. That means it is true only if y<9

Assignment operator

As discussed before, the assignment operator connects the data with a variable name.

Name of operator	*Symbol*	*Example*	*Result*
Assignment equal to	=	x=2	Value of x in memory location becomes 2

Remember that assignment equal to (=) and relational equal to (= =) are with a different meaning.

Precedence of operators

On solving a mathematical expression, the actions of operators are based on some rules of precedence. The computer will scan the expression from left to right. It will execute the first precedence operators in the order of occurrence. Then again scan from left to right for second precedence operators and will execute the second precedence operators in the order of occurrence. This will continue up the final answer. At any instant precedence rule can be violated with the proper use of parenthesis. So special care is to be given in framing a mathematical expression, which is free from semantic error. The following table lists the precedence order from high to law.

1 Logical NOT, Negation
2 Exponentiation
3 Multiplication, Division, Modulus division, Floor division
4 Addition, Subtraction
5 Less than, Less than or equal, Greater than, Greater than or equal
6 Equal, Not equal
7 Logical AND
8 Logical OR
9 Simple assignment

3.2.2 Expressions

A meaningful combination of operators, constants and variables is known as ***expression***. Expressions are classified as follows.

1. **Arithmetic expression**: It is an expression with arithmetic operators, constants and variables. $ut+{}^1/_2 at^2$, 1.28mt/x etc are valid examples. Arithmetic expressions result into a numeric value that can be used directly with an assignment statement or logical expression. The following are examples of some valid logical expressions.

*y+2/3+x**3*

*(y+2)/3+x**3*

*(a+b)*1.28*

2. **Logical expressions**: It is the expressions that result in true or false. A logical expression is a combination of relational operators or logical operators or both operating on variables, constants, and arithmetic expressions. It is used only for condition checking and not fit for the assignment. Go through the following examples.

x > 10

x <=y+3

*(a+b)*1.28 > d and a!=b*

x= = 3

The result of an expression must be rigid and single-valued without any ambiguity. Building up a proper expression will reduce the chance of semantic error. Special care is to be given to minimise the time and memory space during the construction of an expression.

2.15 Initialisation

The selection of data type is automatic in python. So, the initialisation of the variable is not required. If we are doing a re-assignment statement, the variable must be initialised before its first use. If it is not initialised properly, the result may be wrong. Consider the statement.

sum=sum+ term

In this, we are reassigning the value of sum to make a series of additions. So, the initial value of sum must be zero. So, we must start with

sum=0

sum=sum+ term

If it is not initialised, the computer may give some initial value depends on the assembly. It will make a wrong result. Same the case in the instruction x=x*term. Here the initial value of x must be one. No need for any initialisation for the variables associated with the assignment statement like x= a*3 or rad_angle = theta*3.14/180.

2.16 Functions

Large programmes need to be divided into small logical units. A ***function*** is a small, isolated unit of code with a name, and it can perform a definite work. From any part of the programme, we can call this function by name. Then the computer will execute that code to do the defined job. When we are calling a function, we

must pass some data to the function, for its proper working. These data variables are called ***arguments***. On getting the arguments, the function will execute the instructions for doing the prescribed work. After doing it, control will return to the main programme with one or more answers. Functions are divided into ***user defined functions*** or ***built-in functions*** or ***library functions.***

Built-in functions

One of the major advantages of Python is the availability of various libraries for various applications like scientific computation, graphics, networking etc. The standard library is divided into many modules like time, random, pickle, system etc. Each module consists of many built-in functions.

To get the service of such functions, we must instruct the compiler to load the corresponding module or function library to RAM. Following are the examples

```
from math import sin
print(sin(x))
```

Here we are loading the function to calculate sin(x) from the module named math. But with this type of loading, we can calculate only sin(x) using the math library. Other functions defined in the math library will not work. The same thing can do in another way.

```
from math import *
print(sin(x), cos(x))
```

In this example, we are importing the entire mathematical library into RAM. So, we can call any functions defined in that module. There is another method to do the same thing. But there may be an overloading of RAM.

```
import math
print(math.sin(x))
print(math.cos(x))
```

If the library name is large, the same instructions can be in another form.

```
import math as m
print(m.sin(x))
print(m.cos(x))
```

Similarly, there are many functions defined in different libraries.

```
from NumPy import *
from pylab import *
```

List of important mathematical functions

(This is a partial list. A full list is available on the web. To activate these functions,

we must import math library)

ceil(x): Function returns the smallest integer value greater than or equal to *x*. Input x must be float.

x=12.8

y=12.3

ceil(x)=13

ceil(y)=13

floor(x): Function returns the smallest integer value less than or equal to *x*. Input x must be float.

x=12.8

y=12.3

floor(x)=12

floor(y)=12

fabs(x): Function returns the absolute value of *x* as a float. Input x may be integer or float.

fmod(x, y): Function returns the reminder in x/y as a float. Inputs may be integer or float. It is almost identical to x%y. But there may be a slight difference in the accuracy. According to the developing team of Python fmod() is more accurate when working with floats, while x%y is more accurate when working with *integers.*

x=10

y=9

fmod(x,y)=1.0

fsum(x): Function returns a floating point sum of values in the list named x

x=[.1,.2,.3,.4]

fsum(x)=1.0

trunc(x): Function returns the integer part of the float x after truncation.

x=12.473

trunc(x)=12

exp(x): Function returns e**x.

log(x, b): With one argument, function returns the natural logarithm of *x* to the base *e*. With two arguments, it returns the logarithm of *x* to the given base b

log10(x): Function returns the logarithm of *x* base-10. This is usually more accurate than log(x, 10).

pow(x, y): Function returns x raised to the power y.

sqrt(x, y): Function returns the square root of *x* as a float.

acos(x): Function returns the arc cosine of *x*, if x is in radians.

asin(x): Function returns the arc sine of *x*, if x is in radians.

atan(x): Function returns the arc tangent of *x*, if x is in radians.

cos(x): Function returns the cosine of *x*, if x is in radians.

sin(x): Function returns the sine of *x*, if x is in radians.

tan(x): Function returns the tangent of *x*, if x is in radians.

cosh(x): Function returns the hyperbolic cosine of *x*, if x is in radians.

sinh(x): Function returns the hyperbolic sine of *x*, if x is in radians.

tanh(x): Function returns the hyperbolic tangent of *x*, if x is in radians.

degrees(x): Function converts angle *x* from radians to degrees.

radians(x): Function converts angle *x* from degrees to radians.

factorial(x): Function returns factorial of x as an integer. X must be a positive integer.

List of important general functions

For the easiness of programming, python supports many general functions defined in general library. No need of any import statements to access these functions within a programme. This is a partial list. Full list is available in web.

chr(*i*) : Function returns a string of one character whose ASCII code is the integer

chr(97)='a'

help(*object*): Invoke the built-in help system in interactive mode. If no argument is given, the interactive help system starts on the interpreter console. For example, type 'help (input)' in the command prompt. Then the system will show the exact syntax of the instruction as follows.

input(...)

input([prompt]) -> value

len(*s*) : Return the number of items in a list or the number of characters including white space in a string or set.

x=[2,8,3.7,4]

len(x)=4

x='hello'

len(s)=5

list(x) : Return a list of elements in x , if x represents a list

x=[2,8,3.7,4]

list(x)

[2, 8, 3.7, 4]

max(x): If x represents a list or set, function will return the element of highest value.

x=[2,8,3.7,4]

max(x) =8

min(x) : If x represents a list or set, function will return the element of lowest value.

x=[2,8,3.7,4]

max(x) =2

oct(x) : Convert an integer number to an octal string. The result is a valid Python expression.

hex(x): Convert an integer number to a hexadecimal number string. The result is a valid Python expression.

round(*x*, *n*):Return the floating-point value *x* rounded to *n* digits after the decimal point. If *n* is omitted, it defaults to zero.

round(23.458)=23.0

round (23.458,1)=23.5

min(*n1, n2, n3,...*) : If n1, n2,n3 ... are variables representing numbers, function will return the value of the lowest. If n1, n2, n3... are numbers, system will show the lowest number.

x=12

y=15

min(x,y)=12

min(12.8, 25, 2, 14.32)=2

max(*n1, n2, n3,...*): If n1, n2,n3 ... are variables representing numbers, function will return the value of the highest. If n1, n2, n3... are numbers system will the highest number.

x=12

y=15

max(x,y)=15

max(12.8, 25, 2, 14.32)=14.32

User defined functions

A user defined function is a named part of the programme made by the user. It can be considered as a sub-programme, within the main programme. The naming rules of function are the same as that of a variable. It can be invoked from the other parts of the programme as often needed. Let us go through an example. Suppose we want to find the square root and cube root of numeral very often in a program. So, we can incorporate two program segments named sqr and cube as functions. At any instant when we need a square root or cube root, we can ask the control to go to the respective function to find it and return. Go through the following examples.

```
def cube(x)
    return x*x*x
def sqr(x)
    return x*x
num=input('Enter the number')
print ('The square of the number is ', sqr(num))
print('The cube of the number is ', cube(num))
```

'def' is the keyword to make a function definition. 'sqr' and 'cube' is the name of the function. The general syntax of a function definition is the following.

```
def function name (list of arguments to be passed if any)
    function body
    return name of the argument if any
```

The rule for function naming is the same as that of variable naming. For the proper working of a function, we must pass the data to the function from the main programme. The variable using in the function definition is called ***dummy argument***. The number of arguments used in the function call must be the same as that of the function title. The variable name used in both places may be different. In the above example, the variable in a function call is 'num' and that in function title is 'x'. If there is no transfer of data from the main programme to function, we can omit the dummy argument. But the parenthesis along with the function name is compulsory. We can write all calculations and instructions just below the function definition statement with an indent. These lines will form the ***function body***. After executing all instructions in the intended block, control will return to the main programme. During returning to the main programme, if any data is to be transferred, we can use the statement 'return'. The syntax is,

return (list of variables to transfer, separated by comma).

But, the usage of return() is optional. Go through the following example.

Example 6:

```
def sq():
    x= input('Enter the number ')
    print(x*x)
print('Demonstration of function')
sq()
#end
```

In this example, a function named 'sq' is defined in the first three lines. Note that, it is without arguments in function title and return statement. During the execution of the main programme, we can call the function with the instruction sq(). After doing the instructions in the function, the control will return to the same point. The output will be as follows.

```
Demonstration of function
Enter the number 12.4↵
153.76
```

Example 7:

```
def vel(t):
    u=0
    v=u+9.8*t
    return v
def pos(t):
    u=0
    s=0.5*9.8*t*t
    return s
print('Demonstration of function')
t=input('Enter the time interval to find velocity ')
print('Velocity = ',vel(t))
print('Position = ',pos(t))
#end
```

Output

```
Demonstration of function
Enter the time interval to find velocity 1.2□
Velocity =  11.76
Position =  7.056
```

Here we are using two functions named 'vel' to calculate the final velocity and 'pos' to calculate the position of a freely falling body after a time interval of t. From the main programme, we can call these two functions. During the call, the system is transferring the value of t to function by using arguments within the parenthesis. After calculations, the system is transferring the calculated values to the main function by the statement 'return'.

It can be noted that the argument in the function call and the argument in the function title is the same in this example. It is not necessarily to be the same. We can use two different variables for the argument in the function call and function title. But the number of arguments used in the function call and the function title must be the same.

2.17 Local variable and Global variable

A variable defined inside a function is known as a ***local variable***. It is not available outside the function. The same variables can be used in different functions for a different purpose. But if we want, a variable defined in one function can be made available in another function. In such a case that variable is known as the ***global variable***. This can be done by using an instruction 'global'.

Example 8:

```
def read():
    global t,u,g
    g=9.8
    u, t=input('Enter the initial velocity and time interval ').split()
def vel(t):
    v=u+g*t
    return v
def pos(t):
    s=u*t+0.5*g*t*t
    return s
print('demonstration of function')
read()
print('Velocity = ',vel(t))
print('Position = ',pos(t))
#end
```

In this programme, the input instructions are in the function named 'read'. The

values of u, t and g are treated as global. So, it is available for calculations in all other parts of the problem.

2.18 Formatted outputs

On some occasions, the output obtained by the simple print instructions is not suitable for display. So, there are some special methods to customize the view according to our desire. For that, we can incorporate the print statement with % operator and format string. The important format strings are the following.

Format string	Data type	Example	Result
d	integer	%3d	Print as an integer. 3 spaces will be reserved
f	float	%7.3f	Print as a float. 7 spaces, including 3 decimals will be reserved
s	String	%30s	Print as a string. 30 spaces will be reserved.
e	exponential	%6.2e	Print as a floating-point exponential. The mantissa part will be a float. 6 spaces, including 2 decimals will be reserved. The exponent part always will be an integer of width 2.

Normally the number will be with the right justification. So, a positive number as the width will justify the numerical value to the right. A negative number as width stands for a left-justified numeral. %3d stands for right justified integer of width 3, whereas %-3d stands for the left-justified integer of width 3. In the case of strings, positive numbers as width can be used to make left justification and negative numbers for right justification. The data along with the formatted print statement must be a tuple. The number of format strings must match in number with the elements in the tuple. Go through the following segments of print statements to clarify the proper usage.

```
x=12.8
y=2.1
print( x
print('The answer is %7.2f'%(x))
print('The answer is %7d'%(x))
print('The sum of %8.2 and %8.2 will %9.3' %(x,y,x+y))
```

Output

```
12.8
The answer is   12.80
```



```
The answer is   12
The sum of   12.80 and   2.10 will be   14.900
```

This will be highly helpful in printing matrix, table values etc.

2.19 Footnotes and comments

Along with the Python programme, we can write our footnotes and comments for future reference. Any line starting with '#' will come in that category. The computer will ignore these sentences from the compilation. Go through the following examples.

```
# This is a programme for the bisection method
# End of programme
print( x)                    # This is to print the initial guess value
```

2.20 List

The list is an important data type of Python. In a list, we can store many elements under a single variable name. The elements may be of any type. Lists are defined by enclosing the elements inside a pair of square brackets, separated by commas. The individual element may be of any type, even another list.

a=[2,8,14.35, 'Thomas']

In the above statement, 'a' is called a list in python. It is much more flexible than a string. Using instructions, we can change the order of elements, add new elements, and remove an existing element. This property of the list is called ***mutability***. A list is mutable where a string is immutable. The variable name corresponding to each element in the list is a[0], a[1], a[2] etc. Here 0, 1, 2, 3 etc are called ***index***.

Example 9:

```
#example for list manipulation
a=[3284,'Thomas',81.43]
print(a)                          # Print the list as it is
print('roll no ',a[0])            # Print the first element
print('Name  ',a[1])
print('Mark %6.2f'%(a[2]))        # formatted printing with list
# See how the elements can be changed.
a[2]=73.5
print('New mark list')
print('Mark %6.2f'%(a[2]))
#end
```

Output

```
roll no 3284
Name  Thomas
Mark 81.43
New mark list
Mark 73.50
```

In a list, the index of the first element is 0. The statement b=a will not make a new list named 'b'. Instead of physically copying it will connect b with a as a reference. That means, when we change a, b also will change. Go through the following programme to demonstrate this.

Example 10:

```
#Example for list manipulation
a=[3284]
print(len(a))            # To print the number of elements
a.append(78.51)          # To add a new member at the end
a.insert(1,'Antony')     # To insert a new element at second position
print(a)
b=a
print(b)
a[1]='Thomas'            # A correction is made in list 'a'
print(b)
#end
```

Output

```
1
[3284, 'Antony', 78.51]
[3284, 'Antony', 78.51]
[3284, 'Thomas', 78.51]
```

Please note that when a correction is done in the list a, list b is also changing.

Table of functions connected with list:

list.remove (x)	Remove the first item from the list whose value is *x*. If there is no such item, the system will report an error.
list.count(x)	Return the number of times *x* appears in the list.
list.sort()	Sort the items in the list.

list.index(x)	Return the index in the list of the first item whose value is *x*. It is an error if there is no such item
list.reverse()	Reverse the elements of the list
list.append(x)	Add an item to the end of the list
list.insert(i, x)	Insert an item at a given position. The first argument is the index of the element before which x is to be inserted. After insertion, the index of the new element becomes i.
len(a)	Find the number of elements in the list named a.
sum (a,x)	Sum up all the elements in the list 'a' with the value stored in x.
sum(a)	to add all elements of 'a'
x in a	Test wheatear *x* is a member of a. System returns true if x is a member of 'a'.
x not in a	Test wheatear *x* is not a member of a. System returns true, if x is not a member of 'a'.
min(a)	If a represents a list, the function will return the element of the lowest value.
del a[x,y]	Delete all or some of the elements in a list named a. x is the index of the element to remove, and y is the number of elements to remove. del a[0] to remove the first element del a[2:4] to remove 4 elements starting from index 2 del a[:] to remove all the elements in the list to make it empty.
sum (a,x)	Sum up all the elements in the list 'a' with the value stored in x.
a+b	Add list 'a' with list 'b'.
*a*x*	Makes a new list that contains the elements x times by concatenation.

2.21 Set

A set is an unordered collection of distinct objects. Functions related to sets can be used to find differences, intersections, and unions of lists, to check the presence of an element in a list, duplicate elements in a list etc. But sets do not support indexing, slicing, or other sequence-like behaviour. There are two built-in set types, ***set*** and ***frozenset.*** The set type is ***mutable***. The **frozenset** type is immutable.

That means, its contents cannot be altered after it is created; it can therefore be used as a dictionary key or as an element of another set. A set can be created as follows.

set([1, 2, 7.6, 'Thomas'])

A list can be converted into a set as follows.

x=[1,2,3,4]to create a list

y=set(x) to create a set

Table of functions connected with set

len(s)	To find the number of elements in the set named s.
x in s	Test wheatear *x* is a member of s. Returns true if x is a member of s.
x not in s	Test wheatear *x* is not a member of s. Returns true, if x is not a member of s.
s.add(x)	add element x to set s
s.remove(x)	remove x from set s; raises an error if not present
s.union(t)	new set with elements from both *s* and *t*
s.intersection(t)	new set with elements common to *s* and *t*
s.difference(t)	new set with elements in *s* but not in *t*
s.symmetric_ difference(t)	new set with elements in either *s* or *t* but not both

2.22 Range

It is an instruction to make a list of numbers automatically with a specified regular pattern within a range.

To create a list of numbers up to 5 we can use,

>>>x=range(5)

Then system will make a pattern of integers. It is with all the properties of the list, but it is not a list.

The output will be as follows:

x=[0,1,2,3,4]

>>>x=range(5,10)

It can be used to make a list of elements 5 to 9 (10 is the upper boundary which is not included) with an interval of one.

>>>x=range(10,100,5) is to make a list starting from 10, up to 99 with a step size of 5. In general, the range command is with the following syntax.

variable name=range(value of first element, upper limit, step size).

If used with only one argument, that argument is treated as the upper limit. In that case, the first element will be zero and the step size will be 1. If used with two arguments, the arguments are treated as the value of the first element and upper limit. In this case, the step size will be 1. The elements produced by range function will be integers always. To create a list of floats, use the following set of commands.

import numpy

numpy.linspace(0.0, 3.0,num=15)

In this the first argument is the starting number, the second argument is the upper limit and the third one is the number of elements.

2.23 Tuple

Tuples are data structures very similar to lists, but immutable. So we can assume that the tuple is an immutable list. They are generally used for data that should not be edited. A tuple may be created directly or converted from lists. Generally, the tuple is enclosed in parenthesis.

lst=[1,'a',6,3.14]

tup=(1,'a',6,3.14)

Here lst is a list whereas tup is a tuple.

A list can be converted into a tuple as follows,

x=tuple(lst).

Then x become (1,'a',6,3.1400000000)

Tuple can be created from items separated by commas also.

t ='A','tuple','needs','no','parens',25,4.67

Chain assignment can work with tuple.

a= b =1,2

Then a=(1,2) and b=(1,2)

Concatenation will work with tuple. The tuple operations and list operations are almost the same with the following exemptions.

1. Slicing is not permitted in a tuple.
2. No index for the elements in a tuple.
3. No append operator.

A tuple can be used just like a list. Lists can be used to return multiple values from a function.

2.24 Dictionary

A dictionary is an associative array enclosed by a set of curly braces. Each element in a dictionary is with two parts named ***key*** and ***value***. A dictionary contains as many as such "key – value" pair in the following form:

x= {key1 : value1 ,key2 : value2 ,key2 : value3 , . . . }

A value can be accessed by a key. Dictionaries may be created directly or converted from sequences. Dictionaries are enclosed in curly braces {}. For example, a dictionary d created as

d ={'city':'Paris','age':38,(102,165):'Data coordinate'}

A list can be converted into dictionary using:

x=dict(lst)

Consider a list named lst as

lst=[('x1',4),('y1', 12),('x2',6),('y2', 18)]

It can be converted into a dictionary named coordinate as follows:

coordinate=dict(lst)

Then coordinate is a new dictionary.

coordinate ={'y1': 12, 'x2': 6, 'x1': 4, 'y2': 18}

Also, dictionaries can be easily created by zipping two sequences which are list or tuple:

seq1 =('a','b','c','d')

seq2 =[1,2,3,4]

d =dict(zip(seq1,seq2))

Then d is a new dictionary as

d={'a': 1,'c': 3,'b': 2,'d': 4}

The operations on dictionaries are somewhat unique. Slicing is not supported by discretionary.

Consider a dictionary d = {'Item':'Cat','Price':2000,'colour':'White'}. Then,

d.keys() =['Item', 'Price', 'colour']

d.values()=['Cat', 2000, 'White']

In a dictionary, the presence of data can be searched by following instructions:

>>>'cat'in d↵

True

>>>'dog'in d↵

False

We can combine two dictionaries by using the update method of the primary dictionary. Note that the update method will merge existing elements if they conflict. Go through the following interaction:

>>>d ={'apples': 1,'oranges': 3,'pears': 2}↵

>>>ud={'pears': 4,'grapes': 5,'lemons': 6}↵

>>>d.update(ud)↵

Then d become

{'grapes': 5,'pears': 4,'lemons': 6,'apples': 1,'oranges': 3}

To remove the item, grapes, from the above dictionary use

>>>del d['grapes']↵

We can call a value by giving the key

>>>d['apples'] ↵

1

>>>d['grapes']↵

5

Exercise

One-word type questions

1. ------ is the originator of Python
2. Python is coming under the category of ------ software.
3. Name the originator of the free software movement.
4. Name the human readable form of programme.
5. The type-system of python is ------
6. The operational modes of Python are ------ and file mode.
7. Interactive mode is also known as ------
8. In Python, >>> is known as ------
9. A python source code must be with an extension ------
10. The only one special character allowed in the identifier name is ------
11. By the dynamic data type system, the computer will store 10/10 as ------
12. An instruction complex (10) will make a complex number t ------
13. Extracting a part, from a string is known as -----
14. The symbol for the floor division is ------
15. Floor division is to find ------

16. Modulus division is to find -----
17. In python the result of 14.8//3 is -----
18. In Python the result of 24%6 is -----
19. A combination of operators, variables and constants is known as ------
20. An arithmetic progression with a starting value and common difference can be made with ----- instruction.
21. The index of the first element of a list is -----

Short answer type questions

1. What is the meaning of software freedom?
2. Explain the conditions by which software is categorised as free software.
3. Explain the backgrounds of Python, which is responsible for its growth.
4. Why is the interactive mode known as calculator mode?
5. Explain the 'dynamic type system'.
6. What is a file mode?
7. Write the general syntax of the input statement.
8. What is the use of a message board in the input statement?
9. Explain the difference between 'print x' and 'print x,'
10. What is the meaning of case sensitivity? Explain it in the background of Python.
11. Explain the difference between a binary operator and a unary operator.
12. Explain the difference between 5%2 and 5.0%2
13. What is an assignment operator? Illustrate with examples.
14. Explain the difference between x=y and x==y
15. What is the fundamental difference between arithmetic expression and logical expression?
16. In an instruction sum=sum* x, is it compulsory to initialise sum? Explain
17. What is the difference between?

    *from math import sin & from math import **
18. Explain the use of the field specifier *'%f'*.
19. What is the method of writing our own comments in a programme?
20. A list is mutable. What is meant by this comment?
21. What is the difference between *print(a, b)* and *print (a, '\n', b)*
22. Find the mistake and correct the following instruction.


```
print('The sum of %d and %d is %d', x, y, x+y)
```

23. What is a set?
24. Range command can create a list of integers only. How can we create a list with float?
25. Write the output from the following code:

```
A=[2,4,6,8,10]
L=len(L)
S=0
for I in range(1,L,2):
    S+=A[I]
Print("Sum=",S)
```

26. Find the errors from the following program:

```
n=input (Enter total number of elements)
l=range(n)
print (l)
for i in (n);
l[i]=input("enter element")
print "All elements in the list on the output screen"
for i on range(n):
    print l[i]
```

28. Write a function group of a list (list, size) that takes a list and splits it into a smaller list of a given size.

Paragraph type questions

1. Explain the advantage of python over another language.
2. Explain different modes of operation of Python with the advantages of each category.
3. Express the documentation language of addition of 10 and 3 in calculator mode.
4. What are the rules for naming a python file in file mode?
5. What are the advantages of file mode?
6. What are the common elements of a Python programme?
7. Explain the difference between formatted output and non-formatted output with examples.

8. Go through the following code segments,

```
x= 12.84
x=str(x)
x=x+x
```

Find the value of x. Explain the result.

9. Explain truncation effect of integer.
10. Explain the use of *type()* instruction.
11. Explain the string concatenation with examples.
12. What is an assignment statement? Explain the difference between $x=y$ and $y=x$.
13. Explain the difference between input and raw_input.
14. Explain the string slicing.
15. What is an arithmetic operator? Illustrate with examples.
16. What is a relational operator? Illustrate with examples.
17. What is a logical operator? Illustrate with examples.
18. Explain the precedence of operators.
19. Explain the concept of functions and their classifications.
20. What is a module? How it can be loaded in the RAM
21. What are the different methods of loading the built-in function of a module in the RAM?
22. What is an argument in connection with functions?
23. In a function call, transfer of arguments is not compulsory. Comment your answer.
24. Find the mistakes in the following code segment.

```
import math
x=x+30
print sin(x)
```

25. Explain the difference between *fmod(x,y)* and *x%y* with the comments of the developers.
26. Explain the difference between the local variable and the global variable.
27. Explain the role of a global variable in the functions without the return statement.
28. Explain the difference between *%8.2e* and *%8.2f*

29. Explain the difference between list and set.
30. What are the permitted slicing operations in the list?
31. What are the permitted operations on a set?
32. What is the use of the range instruction? Explain the syntax.
33. Explain the advantages and limitations of range command.

Long answer type questions

1. What is a variable? Explain the naming rules and usage with special reference to input and output statements.
2. Explain the data types and rules of automatic type conversions in Python. Explain the need for type conversion with specific examples and the typecasting instructions.
3. Explain the different type of operators and their use in Python.
4. Explain the method of making the user defined function with examples.

Chapter 3
Flow of control

3.1 Introduction

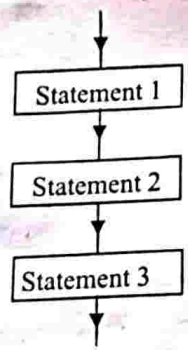
The design of a computer is to execute the instructions sequentially from beginning to end. But in most case, depends on the circumstances, the programme must decide what to do next. For this, we must have the capacity to move the control anywhere, depends on the situation. There are three types of statements to control the flow.

1. Sequential
2. Selective
3. Iteratively

There are special instructions to move the control selectively or iteratively. If nothing specified, control would move sequentially.

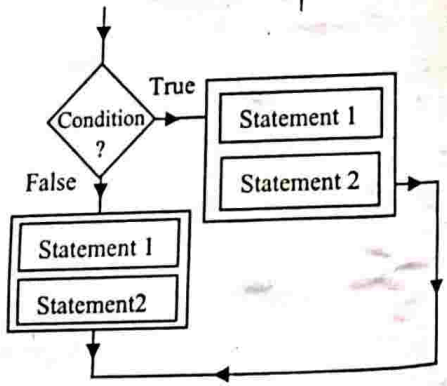
3.1.1 Sequence construct

In sequence construct, the statements are executed sequentially. That means, after executing a statement, the computer will execute the next statement. Then it will execute the next statement. This will continue till the last statement. In a programme, if not instructed otherwise, the execution is sequential.



3.1.2 Selection construct

The selective construct means the execution of statements depending upon a condition test. If the condition results true, a set of statements is followed. Otherwise, another set of statements is followed. The statement using for the selection construct is the conditional statement or decision statement.

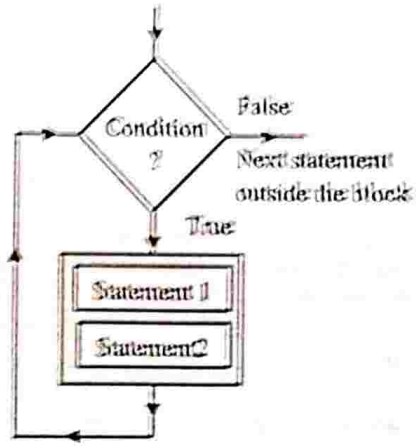


3.1.3 Iteration Construct

In the iteration construct, a set of statements will be repeated depending upon a condition. After checking a

condition, a block of statements is repeated again and again. System stops the iteration when the condition becomes false. The iteration statements also called ***loops***. There are two types of loops in Python. They are ***for loop*** and ***while loop***.

An iteration loop is with four elements.



1. Initialization expression: The flow of each loop is controlled by a variable named control variable. Control variables and other supporting variables are to be initialised before entering the loop.
2. Test expression: It is an expression whose truth value decides wheatear the loop body will be executed or not.

   If the test expression evaluates to false, the loop is terminated.
3. Update expressions

   Update expressions are to change the values of the control variable and the supporting variables after each iteration. With this update, the system will be ready for the next iteration or termination of the loop.
4. Body of the loop

   The body of the loop contains the statements to execute repeatedly. Control can exit the loop either by a normal termination by the failure of test expression or by an abnormal termination using the break statement. But entry to loop is restricted by the execution and passing of test expression.

3.2 Selection statements

There are many methods to make selective construct.

3.2.1 Simple if . . . else

The if...else statement tests a particular condition and if the condition evaluates to true, a course of action is followed. That means the computer will execute a single statement or a block of statements written with a particular indent (right shift). If the condition evaluates to false, another course of action is followed. The syntax of the if...else is the following,

if (conditional expressions with arithmetic or logical operators):

instruction 1

```
nstruction 2
...
lse:
    instruction 3
    instruction 4
    ...
```

In this case, the control will check the condition. If the answer is true, control will execute the instructions in the next block separated by indent. That means instructions 1 and 2. If it is false, control will skip that block to execute the block just below the else statement. That means instructions 3 and 4. The else part is optional. If the else part is not present, the computer will execute the next instruction which is in line with the 'if' statement (inline means, statement with the same indent). The condition statement is a simple relation or a compound relation with relational operators. Associated with a relational operator, there may be logical and mathematical expressions.

Example 1:

Write a programme to find the biggest number from the three given number.

Answer :

```
#programme to find the biggest out of three
x=float(input('Enter the first number '))
y=float(input('Enter the first number '))
z=float(input('Enter the first number '))
big=x
if y > big:
  big=y
if z > big:
  big=z
print ('The biggest number is ', big)
#end
```

xample 2:

'rite a program to find wheatear the typed year is a leap or not, using a 'simple structure.

iswer:

```
#programme to check whether the given year is a leap year
```



```
x=float(input('Enter the year '))
if x%4==0 and x%400 != 0:
  print("%d is a leap year"%(x))
else:
  print("%d is not a leap year"%(x))
print('Thank you')
#end
```

Example 3:

Make a program to find the area of different shapes with a selection menu using simple if structure.

Answer:

```
#programme to calculate the area
from math import sqrt
print(' Choice Menu ')
print(' 1.. Circle')
print(' 2.. Rectangle')
print(' 3.. Triangle')
ch=int(input ('Enter your choice--> '))
if ch==1:
  rad= float(input('Enter the radius  '))
  print('The area = ',3.14*rad*rad)
if ch==2:
  leng=float(input(' Enter the length  '))
  bred=float(input(' Enter the breadth  '))
  print(' The area = ' ,leng*bred)
if ch==3:
  a=float(input('Enter the first side   '))
  b=float(input('Enter the second side  '))
  c=float(input('Enter the third side   '))
  s=(a+b+c)/2
  area=sqrt(s*(s-a)*(s-b)*(s-c))
  print(' The area = ',sqrt(s*(s-a)*(s-b)*(s-c)))
#end
```

3.2.2 Nested if

Nested if is an 'if…else' statement within another 'if…else' statement. The second 'if…else' can be made in the 'if' block or 'else' block or in both. We can accommodate 256 (depends on processor and version) such nested if loops within an 'if' block and 'else' block. But as the nesting increases, the complexity of the programme also increases. The selection of an else statement, or an if statement, depends on the indenting. It is with the following syntax form.

```
if condition expression 1 :
   instruction 1
   instruction 2
   ...
   if condition expression 2 :
      instruction 3
      instruction 4
   ...
   else:
      instruction 5
      instruction 6
      ...
else :
   instruction 7
   instruction 8
   if condition expression 2 :
      instruction 3
      instruction 4
   ...
   else:
      instruction 5
      instruction 6
      ...
```

Example 4:

Write a programme to find the grade of a student according to the following pattern with nested if.

90% and above A

80 to 89 B
70 to 79 C
60 to 69 D
50 to 59 E
All others- No grade

Answer:

```
#programme to find the grade of a student
mark=input ('Enter the mark ')
if mark >= 90:
   grade='A'
else:
   if mark >= 80:
      grade='B'
   else:
      if mark >= 70:
         grade='C'
      else:
         if mark >= 60:
            grade='D'
         else:
            if mark >= 50:
               grade='E'
            else:
               grade='No grade'
print 'Grade :',grade
print 'Thank you'
#end
```

Example 5:

Write a programme to find the solution of a quadratic equation, with nested loop for different cases.

Answer:

```
#Solution of quadratic equation
from math import sqrt
```

```
a,b,c=input("Enter the Coefficients a, b and c separated by space ").split()
a,b,c=float(a),float(b),float(c)
if a==0:
  print("Equation is not quadratic ")
else:
  dis=b*b-4*a*c
  if dis > 0:
    root1=(-b+sqrt(dis))/(2*a)
    root2=(-b-sqrt(dis))/(2*a)
    print('Roots are real and distinct')
    print('The roots are ',root1,' and ', root2)
  else:
    if dis==0:
      root1=-b/(2*a)
      print(" The roots are real and same")
      print('The single root is ',root1)
    else:
      dis=-dis
      x=b/(2.0*a)
      y=sqrt(dis)/(2*a)
      root1=complex(x,y)
      print(" The roots are real and same")
      print('The single root is ',root1)
print('Thank you')
#end
```

3.3.3 Ladder if

It is a common programming construct that is very easy to handle. The expressions are evaluated from top to downward. The system will go through the first condition. If the answer is true, the immediate block will be executed. Then the system will not check for the next conditions. If the answer is false, the system will go through the next condition. If the answer to that condition is true, the immediate block will be executed. Then the system will not check for the next conditions. If the answer is false, the system will go through the next condition. This will continue. It is with the following syntax form.


```
if condition expression 1 :
   instruction 1
   instruction 2
   ...
elif condition expression 2 :
   instruction 3
   instruction 4
   ...
elif condition expression 3 :
   instruction 5
   instruction 6
   ...
else :
   instruction 7
   instruction 8
   ...
```

Example 6:

Rewrite the example 4 with ladder if.

Answer:

```
#programme to find the grade of a student
mark=input('Enter the mark ')
mark=float(mark)
if mark >= 90:
  grade='A'
elif mark >= 80:
  grade='B'
elif mark >= 70:
  grade='C'
elif mark >= 60:
  grade='D'
elif mark >= 50:
  grade='E'
else:
```

```
    grade='No grade'
  print('Grade :',grade)
  #end
```

Example 7:

Rewrite the example 5 with ladder if.

Answer:

```
  #Solution of quadratic equation
  from math import sqrt
  a,b,c=input('Enter the Coefficients a,b and c separated by space ').split()
  a,b,c=float(a),float(b),float(c)
  if a==0:
    print('Equation is not quadratic')
  else:
    dis=b*b-4*a*c
    if dis > 0:
      root1=(-b+sqrt(dis))/(2*a)
      root2=(-b-sqrt(dis))/(2*a)
      print('Roots are real and distinct')
      print('The roots are ',root1,' and ', root2)
   elif dis==0:
      root1=-b/(2*a)
      print('The roots are real and same')
      print('The single root is  ',root1)
    else:
      dis=-dis
      x=b/(2.0*a)
      y=sqrt(dis)/(2*a)
      root1=complex(x,y)
      root2=complex(x,-y)
      print('Roots are imaginary and distinct')
      print('The roots are ',root1,' and ', root2)
#end
```


3.4 Iteration Statements

There are many methods for making iteration construct.

3.4.1 While loop

At the entry of this loop, a particular condition is evaluated with the test expression. If the expression evaluates to true, a course of action is followed. The statements in the action (loop body) must be written with intent. Within the loop body, control variable and supporting variables can be updated. Again, the control will go to the test expression. If the expression evaluates to true again, the course of action will repeat with new data. This will continue till the test expression evaluates to false. The syntax of the while loop is,

while condition expression:
instruction 1
instruction 2
. .
. .

Example 8:

Write a programme to print the multiplication table for any given number.

Answer:

```
  #programme to print multiplication table
  x=input('Enter the quantity to make table  ')
  x=int(x)
  print('Multiplication table of ',x)
  n=1
  while  n<=12:
    print(' %3d  X  %3d  = %3d'  %(n,x,n*x))
    n=n+1
  #end
```

Example 9:

Write a programme to print first n natural numbers and to print their sum

Answer:

```
  #programme to print first n natural numbers and their sum
  x=input('Enter the number  ')
  x=int(x)
```

```
print('The list of natural numbers up to ',x)
n=1
sum=0
while n<=x:
  print(n,end=(' '))
  sum=sum+n
  n=n+1
print('\n The sum of natural numbers up to %d is %d' %(n, sum))
print('Thank you')
#end
```

Example 10:

Write a programme to check whether the given number is a prime or not

Answer:

```
#programme to check the prime number
x=input('Enter the number ')
x=int(x)
flag=0
if x%2 ==0:
  print(x,' is not a prime number')
  print('It is divisible with', 2)
else:
  n=3
  while n<x:
    if x%n==0:
      print(x,' is not a prime number')
      print('It is divisible with ', n)
      flag=1
      break
    n=n+2
  if flag != 1:
    print(x,' is a prime number')
print('Thank you')
#end
```


Note: Please note the usage of the break instruction for the termination of the loop. The instruction 'while 1' is to make a loop with 'entry to all' and 'iterate for ever'. To exit from such a loop, we can use break instruction.

3.4.2 For loop

For loop is another method of looping in Python. The general syntax of the 'for loop' is the following:

for name of control variable in name of list containing different values for the control variable:

instruction 1

instruction 2

...

Eg:
```
for i in x:
    f=factorial(i)
    print('factorial of %d is  %f ' %(i, f))
```

First, we must construct a list named x (can give any name) by any method. Here 'i' is called as the control variable. During the execution of the body of the loop, the system will assign the first data element to i. With this value, the instructions in the body will work. At the end of the body of the loop, again the control goes to the top. Then the system will assign the value of the second element to i. With this value, the computer executes the instructions in the block once again. This process repeats for all elements. Consider a list named num with following elements.

num= [1,3,7,4,2,6]

Write an instruction - *for int_no in num:* - followed by the body of the loop.

Control will enter into the body of the loop with int_no=1. After executing all instructions in the block, control will move to the top, again enter into the body of loop with int_no=3. After executing all instructions once again, int_no become 7. This will continue for all values up to 6. Then the loop terminates.

Rules

1. At any stage, we can quit from the loop with the 'break' command.
2. We can use the data stored with the control variable for any purpose. But we cannot change the value of the control variable using any instruction.
3. Getting into the body of the loop is only through the 'for' command. That means we cannot start execution from the middle of the block straight away using jump statements.
4. The control variable may be float or integer with any name. An integer list can

be made with range instruction as discussed in the last chapter. To get the list named num, we can use,

num= range(1,6,1)

For a clear understanding go through the following example to print the multiplication table of 8

```
x=input('Enter the number to make the table  ')
y=range(1,11)
for i in y:
    print ('%d  X  %d  =  %d' %(i, x, i*x)
```

In this example range command will create a list named y with elements 1, 2, 3...10. The third element in the range command is omitted since the step value is 1. In the first phase, the value of 'i' becomes 1. Let the value of x is 8. Then the system will print

1 X 8 = 8

After the printing value of i become 2. With this value, the system will print

2 X 8 = 16

This will continue up to i=10. The printout corresponding to this is

10 X 8 = 80

For convenience, we can mix the for command with range command of the above example as

```
for i in range (1,10)
    print ('%d  X  %d  =  %d' %(i, x, i*x)
```

Example 11:

Write a programme to find the factorial of a number from fundamentals.

Answer:

```
#programme to find factorial
x=input('Enter the number  ')
x=int(x)
fact=1
lst=range  (x,0,-1)
for i in lst:
   fact=fact*i
print('The factorial of ', x , ' is  ', fact)
print('Thank you')
```



```
#end
```

Example 12:

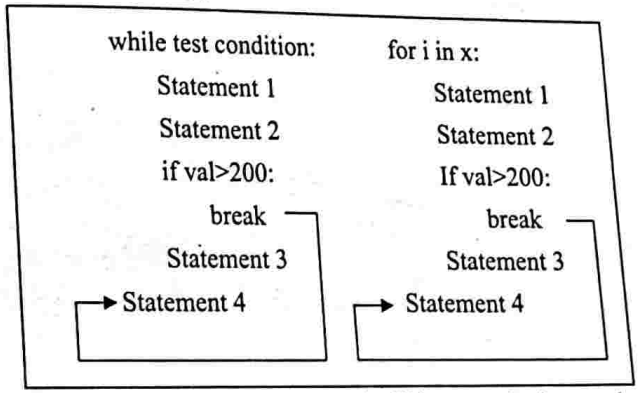
Write a programme to find the product of 10 alternate natural numbers starting from the number entered from the keyboard using for loop.

Answer:

```
num=input('Enter the starting number  ')
num=int(num)
prod=1
for x in range (num, num+10,2):
   prod=prod*x
print('The product is', prod)
print('Thank you')
#end
```

3.5 Break

A break statement is a ***jump statement*** in python to terminate the innermost loop by skipping the reaming part.



After the termination of the loop, control will jump to the instruction next to the body of the innermost loop. In any loop, the break statement will terminate the loop unconditionally. Go through the following code segment.

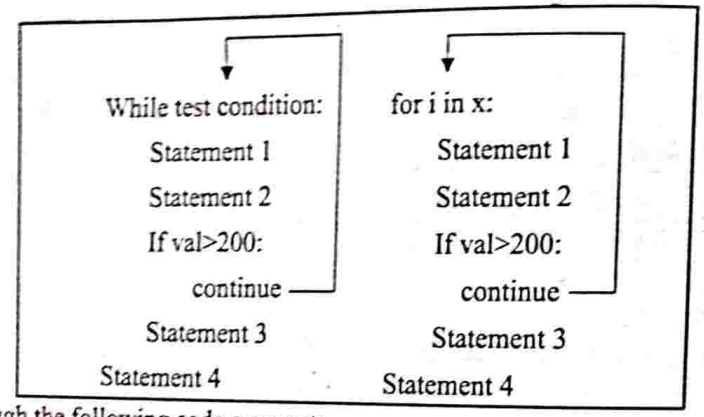
```
#example for a programme with exit code:
x=input('Enter the dividend. 0 to quit  ')
x=int(x)
if x==0:break
```

```
y=input('Enter the divisor  ')
x=float(x)
y=float(y)
print('The result is ',x/y)
print('Programme over, Thank you')
```

In the above code, using the first statement, the control can access the infinite loop. Inside the loop, control asks for the first number and then the second number to calculate the fraction. Then again control will ask for the next set of number due to the presence of an unconditional while loop. This is a problem with the ***exit*** code. Zero is the exit code. When the user types zero for the dividend, control gets inside the loop and then break the loop. Because of break instruction, control jumps to the last statement, which is outside the body of the loop.

3.6 Continue

It is another ***jump statement*** to skip the remaining part of the loop for that iteration. After skipping that part, control will go up for the next iteration.



Go through the following code segment:

```
x=1
while x==1:
  rol=input('Enter the Rollno  ')
  rol=int(rol)
  if rol==0:
    print('Error in keyboard entry. Retype the data')
    continue
  mark=input('Enter the mark out of 80  ')
```



```
  mark=float(mark)
  if mark > 80 :
    print('Error in keyboard entry. Retype the data')
    continue
  print('The percentage of mark is ',mark *100/80.0)
  x= input('To continue, type 1 or any number to exit ')
  x=int(x)
print('Programme over, Thank you')
#end
```

In this programme, when the operator types 0 as roll no, control jumps into the 'if' structure. There, it gets the instruction 'continue' just after the error message. Then control moves to the top by skipping the rest. Again, it will ask for roll no. Similarly, in the case of mark entry, the system skips the rest, if the typed mark is greater than the maximum. The above programme will be more attractive, with the following modifications. Note the changes and their effects.

```
x=1
while x==1:
  rol=input('Enter the Rollno  ')
  rol=int(rol)
  if rol==0:
    print('Error in keyboard entry. Retype the Rollno')
    continue
  while 1:
    mark=input('Enter the mark out of 80  ')
    mark=float(mark)
    if mark > 80 :
      print('Error in keyboard entry. Retype the mark')
      continue
    else:
      break
  print('The percentage of mark is ',mark *100/80.0)
  x= input('To continue, type 1  ')
  x=int(x)
```

```
    print('Programme over, Thank you')
    =end
```

In the nested loop, continue will skip the rest of the commands of that particular loop only.

3.7 Exit

This is a built-in function defined in the module sys. Using this we can terminate the entire program at any stage. The syntax is,

```
import sys
sys.exit()
```

3.8 File Operations

Use the file instructions, we can import data from a file or export data to a file. For both operations, we must open the file at the beginning. A file can be opened in two modes: ***Read mode(r)*** and ***write mode(w).*** During the opening, we must give a dummy name for the file, which is valid inside the programme. To open a file named 'mark.dat' with a dummy name m1 in 'read' mode, the instruction is

```
m1=open('mark.dat', 'r')
```

If opened in the read mode, the file becomes read-only. We cannot write any data into this file. To open in write mode

```
m1=open('mark.dat', 'w')
```

This instruction creates a new file named mark.dat, which is writable. If there is an existing file in this name, it will be deleted. After the use, the file must be closed using *m1.close()*

The general syntax is *Dummy name*=open('*actual file name with extension*', '*mode*') and *Dummy name*.close()

This method can be used to read or write strings only. On reading, if we want, the string can be converted into a number. Go through the following code segment.

```
m1=open('mark.dat', 'w')
m1.write('Mark list')
m1.close()
```

By using this code segment, we can open the file with a dummy name m1. Then a string can be written in the file. The last instruction is to close the file. To read a string from a file the following code can be used.

```
m1=open('mark.dat', 'r')
x=m1.read()
print(x)
```



```
m1.close()
```

We can read or write line by line also. To enter marks of five students which is numeric, we can use the following instructions.

```
m1=open('mark.dat', 'w')
m1.write('Mark list')
for I in range(1,6):
    ma=input("Enter the mark  ")
    ma=float(ma)
    s='\n%3d' %(ma)
    m1.write(s)
m1.close()
```

The content of the file created will be as follows,

Mark list
34
45
37
23
42

During the time of reading, we can convert the marks into numeric.

```
m2=open('data.txt', 'r') # To open the file in read mode
mark_lst=[]              # To create  a list to store data
while 1:
    s=m2.readline()      # To read first line
    if s==":             # To check the end of file
        break
s=int(s)                 # To convert data in one line to an integer
mark_lst.append(s)       # To add the data to the list
#end of data reading
Print( mark_lst)
m2.close()
```

In the above example, using a while loop, data can be read line by line. For further action, it can be converted into the required type by the casting operator. The end of the file will return a black character, which can be tested by the 'if

condition' for terminating the loop.

In all the above methods, data storage is in the form of strings. To store other data types, lists, dictionaries etc, provisions are in the module named pickle.

3.9 Pickling

Strings can easily be written to a file and read from a file as discussed in the previous section. But numbers take more effort since the read() method returns Strings, which have to be converted into a number by the user. saving and restoring data types like lists, dictionaries etc. are very complicated in this mode. To solve this issue Python provides a standard module called pickle.

Pickle has two main methods. The first one is 'dump', which dumps any complex object to a file object without losing its properties and internal structure. It is by serializing and converting the object into a set of binary data. The instruction *pickle.dump(num,fle)* will save any type of object with the name '*num*' to a file opened as '*fle*'. Here '*num*' stands for any data type like list, tuple, dictionary etc.

The second module is 'load', which loads back the object from a file without losing its structure. This is also known as unpickling. The instruction dta=*pickle.load(fle)* retrieves data from the file opened as 'fle' into the identifier 'dta' in the same format.

These instructions are defined in the module pickle.py So we must import this module first. The following code segment can be used for writing into a file.

```
import pickle
m1=open('mark.dat', 'w')
mak=[24,45,32,41,25]
# Here mark.dat is the file opened with a dummy name m1 in write mod and
#                                                    mak is a list
pickle. dump(mak,m1)
m1.close()
```

To read from the file the following code can be used.

```
import pickle
m2 =open('mark.dat', 'r')
x=pickle.load(m2)
#x will be the list already stored in the file mark.dat
Print( x)
m1.close()
```


Example 13:

Write a programme to find the velocity of a free-falling body under gravity at different time intervals, up to a maximum velocity entered from the keyboard.

Answer:

```
vf=input('Enter the maximum value of velocity  ')
vf=float(vf)
dt=input('Time interval ')
dt=float(dt)
t=v=0
print( ' Time       Velocity ')
print( ' %7.3f     %7.3f' %(t,v))
t=t+dt
while v <= vf:
  v=9.8*t
  print( ' %7.3f     %7.3f' %(t,v))
  t = t+dt
print('Thank you')
#end
```

Example 14:

Write a programme to generate the list of all prime numbers in between two given numbers.

Answer:

```
#programme to find the prime numbers
lo=input('Enter the lower number  ')
up=input('Enter the upper number  ')
lo=int(lo)
up=int(up)
a=[]
print('List of Prime numbers within the range\n\n')
for x in range(lo,up):
  flag=0
  if x%2 ==0:
    continue
```

```
    for n in range(3,x,2):
        if x%n==0:
            flag=1
            break
    if flag==0:
        print(x,end=' ')
print('\nThank you')
#end
```

Example 15:

Write a programme to generate Fibonacci series up to a limit.

Answer:

```
#programme to find the Fibonacci series
up=input('Enter the upper number limit  ')
up=int(up)
a=[]
print('List of Fibonacci numbers up to ', up)
x1=1
x2=1
a.append(x1)
a.append(x2)
while x1+x2 <=up:
    x3=x1+x2
    a.append(x3)
    x1=x2
    x2=x3
for k in range(0,len(a)):
    print(a[k],end=' ')
print('\nThank you')
#end
```

Example 16:

Write a programme to solve the following series for n terms

$$y = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \cdots$$

Answer:

```
#programme to evaluate the series
from math import *
x=input('Enter the  value of x      ')
x=float(x)
num=input('Enter the  number  of terms ')
num=int(num)
sum=x
for k in range (3, num+1,2):
    sum=sum+pow(x,k)/factorial(k)
print('Value of the series is', sum)
print('Thank you')
#end
```

Exercise

One-word type questions

1. If nothing is specified, the control flow in python is -------
2. Loop is the alternate name of --------- construct.
3. ------- is the instruction for the abnormal termination of a loop in Python.
4. ------ is the instruction for the abnormal termination of a programme in Python.
5. ------- is the instruction to skip few lines of a loop in Python.
6. -------- is a jump instruction to get outside the loop, after terminating the loop.
7. Exit instruction is to exit from --------
8. -------- is a list of immutable elements.
9. Normally a file can store all data in ---------- form only.

Short answer type questions

1. Write a programme to print 2,4,8,16,32,64,128 using the module math.
2. Given the following code fragment of Python.

```
If a==1:
    print('One')
if a==2:
    print('Two')
```

```
if a==3:
    print('Three')
```

3. Rewrite the above segment using ladder if.
4. What is the meaning of flow control? What are the different flow controls?
5. Explain sequential construct.
6. Explain selective construct.
7. Explain the iterative construct.
8. What is the syntax of the while loop?
9. What is the meaning of iteration in computer language?
10. Name the different elements of an iteration loop.
11. What is the syntax of for loop?
12. Clearly distinguish between the 'assignment equal to' and 'conditional equal to' with respect to a loop.
13. What is the syntax of the 'if else' block?
14. What is the syntax of nested if?
15. What is the syntax of 'ladder' if?
16. Explain the major difference between tuple and list.
17. Explain the role of a tuple in the function sub-programmes.
18. What is a dictionary? Write an example.
19. Explain zip instruction.
20. How we can separate key and value from a dictionary.
21. What is the difference between the 'index' of a list and the 'key' of a dictionary?
22. Explain the difference between the file name and the dummy file name.
23. Explain the difference between dump and write.
24. Explain the difference between load and read.
25. Explain the difference between 'read' and 'readline'.

Paragraph type questions/Programmes

1. Explain the different elements of an iteration loop.
2. Explain the different types of selection statements in python with clear syntax and example.
3. Explain the working mechanism of a conditional statement with the 'if' loop.
4. Explain the working of any two jump instructions common in Python.

5. Explain the read and write operations in a file.
6. With relevant examples, explain the method of reading and writing numeric data in a file.
7. Is the reading from a file is line by line or character by character? Explain with example.
8. Explain the difference in file manipulation with or without pickle.
9. What is the role of 'end of file' in a programme? How it can be achieved?
10. Write a programme to calculate the area and volume of a box with sides a, b, and c. Given that area =2(ab+bc+ca) and Volume=a*b*c. The programme must be continuous with an exit code.
11. A projectile is fired at an angle that varies between 30 to 60 in steps of 5. Make the tabulated values of the range. u=10, g=9.8. Range $= \frac{u^2 \sin 2\theta}{g}$
12. Four resistances are connected in series, write a programme to find the total resistance.
13. Write a programme to find e^x by solving the series up to n terms.

$$e^x = 1 + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

14. Write a programme to convert temperature in Fahrenheit to Celsius or Celsius to Fahrenheit upon user's choice, with an exit code.
15. Write a programme to check whether the given number is a palindrome.
16. Write a programme to find whether the given number is odd or even or prime. The programme should continue as long as the user wants.
17. Write a programme to evaluate the following series up to the desired accuracy level.

$$cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$$

18. Write a program to print out the sum of all odd numbers and sum of all even numbers, from a series of numbers entered through the keyboard continuously. The entry terminates when the number entered is zero.
19. Write a program to find the largest, odd number and largest even number from a series of numbers entered through the keyboard continuously. The entry terminates when the number entered is zero.
20. Write a programme to find the wavenumber and series name of a hydrogen spectral line by entering the shell number of transitions.

Long answer type questions

1. Explain different flow controls in Python and equivalent instructions with syntax and example.
2. Explain the difference in the working of a ladder if and nested if with examples.
3. With syntax rules explain the difference between the 'break' statement and 'continue' statement. Demonstrate it with an example.
4. Make a comparative study between 'for loop' and 'while loop' with example.
5. Write a programme for solving the quadratic equation with nested if and rewrite it with ladder if.
6. Write a programme for printing the multiplication table of a given number with for loop and rewrite it with while loop.